Healthcare.

Course-end Project 2

DESCRIPTION

Problem Statement NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly, and consequential diseases. The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Build a model to accurately predict whether the patients in the dataset have diabetes or not.

Dataset Description:-

The datasets consists of several medical predictor variables and one target variable (Outcome). Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and more.

Variables Description:- Pregnancies- Number of times pregnant, Glucose Plasma- glucose concentration in an oral glucose tolerance test, BloodPressure- Diastolic blood pressure (mm Hg), SkinThickness- Triceps skinfold thickness (mm), Insulin- Two hour serum insulin, BMI- Body Mass Index, DiabetesPedigreeFunction- Diabetes pedigree function, Age- Age in years, Outcome Class variable (either 0 or 1). 268 of 768 values are 1, and the others are 0

Project Task: Week 1 Data Exploration:

1. Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value:

• Glucose

• BloodPressure

• SkinThickness

• Insulin

• BMI

2. Visually explore these variables using histograms. Treat the missing values accordingly.
3. There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

Project Task: Week 2 Data Exploration:

1. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.
2. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.
3. Perform correlation analysis. Visually explore it using a heat map.

Project Task: Week 3 Data Modeling:

1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.
2. Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.

Project Task: Week 4

Data Modeling:

1. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used.

Data Reporting:

2. Create a dashboard in tableau by choosing appropriate chart types and metrics useful for the business. The dashboard must entail the following:

a. Pie chart to describe the diabetic or non-diabetic population

b. Scatter charts between relevant variables to analyze the relationships

c. Histogram or frequency charts to analyze the distribution of the data

d. Heatmap of correlation analysis among the relevant variables

e. Create bins of these age values: 20-25, 25-30, 30-35, etc. Analyze different variables for these age brackets using a bubble chart.

```
# import the libararies
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

```
# import the dataset
df_train = pd.read_csv('health care diabetes.csv')
```

```
# Data Exploration
df_train.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
df_train.columns
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

```
df_train.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

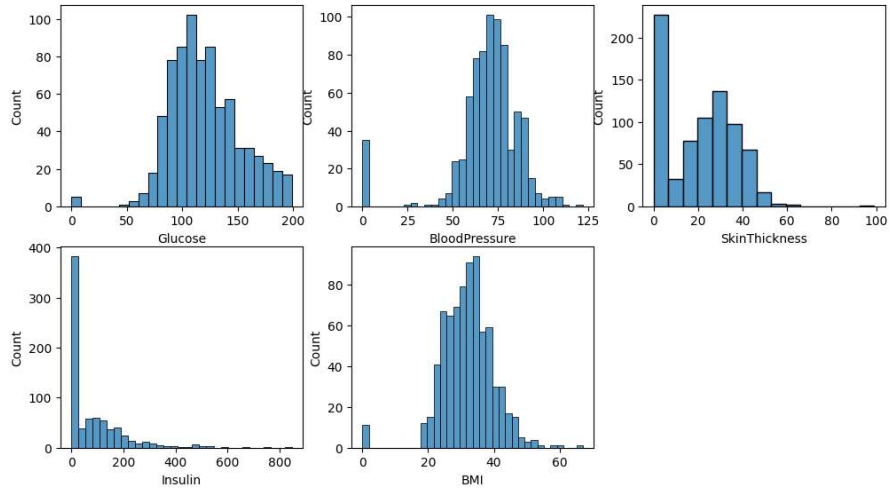
Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value:

• Glucose

- BloodPressure
- SkinThickness
- Insulin
- BMI

```
# plot the histogram to check the values in the column
plt.figure(figsize=(12,10))
plt.subplot(3,3,1)
sns.histplot(df_train['Glucose'])
plt.subplot(3,3,2)
sns.histplot(df_train['BloodPressure'])
plt.subplot(3,3,3)
sns.histplot(df_train['SkinThickness'])
plt.subplot(3,3,4)
sns.histplot(df_train['Insulin'])
plt.subplot(3,3,5)
sns.histplot(df_train['BMI'])
```

```
<Axes: xlabel='BMI', ylabel='Count'>
```



We can see that we have too many missing values(0) in our dataset.

so, these missing values should be treated with Mean,Median or Imputers.

```
df_train_mean=df_train.copy()
```

```
# selecting all the feature columns to perform the missing value treatment
feature_column_mean = df_train_mean[['Glucose', 'BloodPressure','BMI','SkinThickness', 'Insulin']]
```

The Nan values can be replaced with mean or median or imputer so we'll try all three and see which one fill the values to give balanced dataset.

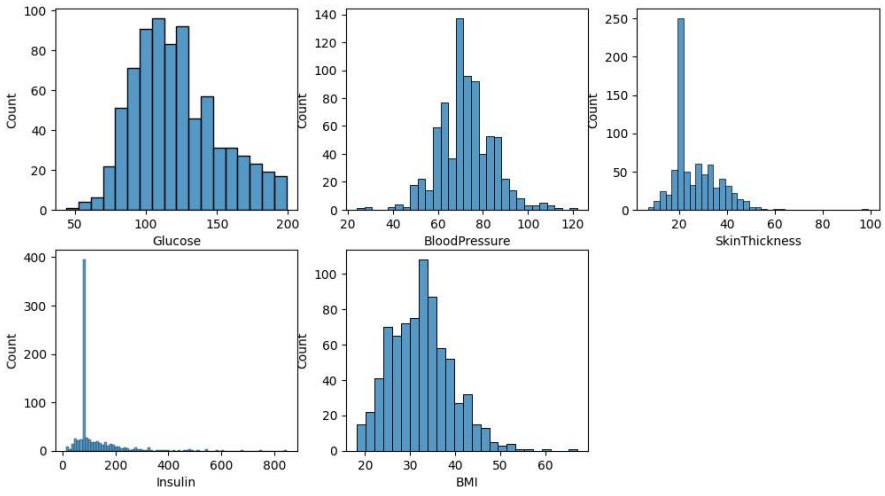
```
# replace the 0(missing value) value with Mean
for i in feature_column_mean:
  df_train_mean[i] = df_train_mean[i].replace(0, df_train_mean[i].mean())
  print(df_train_mean[i].head())
```

```
   0    148.0
   1     85.0
   2    183.0
   3     89.0
```

```
4    137.0
Name: Glucose, dtype: float64
0    72.0
1    66.0
2    64.0
3    66.0
4    40.0
Name: BloodPressure, dtype: float64
0    33.6
1    26.6
2    23.3
3    28.1
4    43.1
Name: BMI, dtype: float64
0    35.000000
1    29.000000
2    20.536458
3    23.000000
4    35.000000
Name: SkinThickness, dtype: float64
0     79.799479
1     79.799479
2     79.799479
3     94.000000
4    168.000000
Name: Insulin, dtype: float64
```

```
# plot the histogram to check the values in the column after replacing missing value with mean
plt.figure(figsize=(12,10))
plt.subplot(3,3,1)
sns.histplot(df_train_mean['Glucose'])
plt.subplot(3,3,2)
sns.histplot(df_train_mean['BloodPressure'])
plt.subplot(3,3,3)
sns.histplot(df_train_mean['SkinThickness'])
plt.subplot(3,3,4)
sns.histplot(df_train_mean['Insulin'])
plt.subplot(3,3,5)
sns.histplot(df_train_mean['BMI'])
```

```
<Axes: xlabel='BMI', ylabel='Count'>
```



It is clean that the dataset is now highly imbalanced. So, let's check for median.

```
df_train_median=df_train.copy()
```

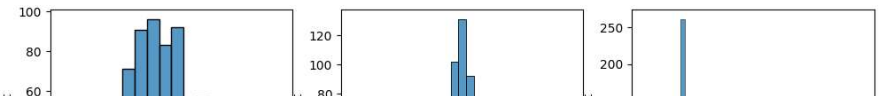
```
# selecting all the feature columns to perform the missing value treatment
feature_column_median = df_train_median[['Glucose', 'BloodPressure','BMI','SkinThickness', 'Insulin']]


# replace the 0(missing value) value with Median
for i in feature_column_median:
  df_train_median[i] = df_train_median[i].replace(0, df_train_median[i].median())
  print(df_train_median[i].head())
```

```
0    148
1     85
2    183
3     89
4    137
Name: Glucose, dtype: int64
0    72
1    66
2    64
3    66
4    40
Name: BloodPressure, dtype: int64
0    33.6
1    26.6
2    23.3
3    28.1
4    43.1
Name: BMI, dtype: float64
0    35
1    29
2    23
3    23
4    35
Name: SkinThickness, dtype: int64
0     30.5
1     30.5
2     30.5
3     94.0
4    168.0
Name: Insulin, dtype: float64
```

```
# plot the histogram to check the values in the column after replacing missing value with median
plt.figure(figsize=(12,10))
plt.subplot(3,3,1)
sns.histplot(df_train_median['Glucose'])
plt.subplot(3,3,2)
sns.histplot(df_train_median['BloodPressure'])
plt.subplot(3,3,3)
sns.histplot(df_train_median['SkinThickness'])
plt.subplot(3,3,4)
sns.histplot(df_train_median['Insulin'])
plt.subplot(3,3,5)
sns.histplot(df_train_median['BMI'])
```

```
<Axes: xlabel='BMI', ylabel='Count'>
```

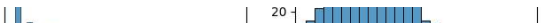


- it is clear missing values filled with median is also giving an imbalanced dataset. So, let's fill missing values usin KNN imputer.

```
# make a copy of train dataframe
df_train_KNN_Im = df_train.copy()
```

```
df_train.columns
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```



```
# select the features where we want to fill the missing value
feature_column_KNN_Im = df_train_KNN_Im[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age']]


# replacing the value '0' with 'Nan'
feature_column_KNN_Im.replace(0, np.nan, inplace=True)
feature_column_KNN_Im.head()
```

```
<ipython-input-74-f09297ba90bb>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing
  feature_column_KNN_Im.replace(0, np.nan, inplace=True)
```

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	148.0	72.0	35.0	NaN	33.6	0.627	50
1	85.0	66.0	29.0	NaN	26.6	0.351	31
2	183.0	64.0	NaN	NaN	23.3	0.672	32
3	89.0	66.0	23.0	94.0	28.1	0.167	21
4	137.0	40.0	35.0	168.0	43.1	2.288	33

```
from sklearn.impute import KNNImputer
# Create a KNN imputer object with k=5
imputer = KNNImputer(n_neighbors=5)

# Perform KNN imputation on the dataset
df_imputed = pd.DataFrame(imputer.fit_transform(feature_column_KNN_Im), columns=feature_column_KNN_Im.columns)


# chek for null value in the new dataframe
df_imputed.isnull().sum().any()
```

```
False
```

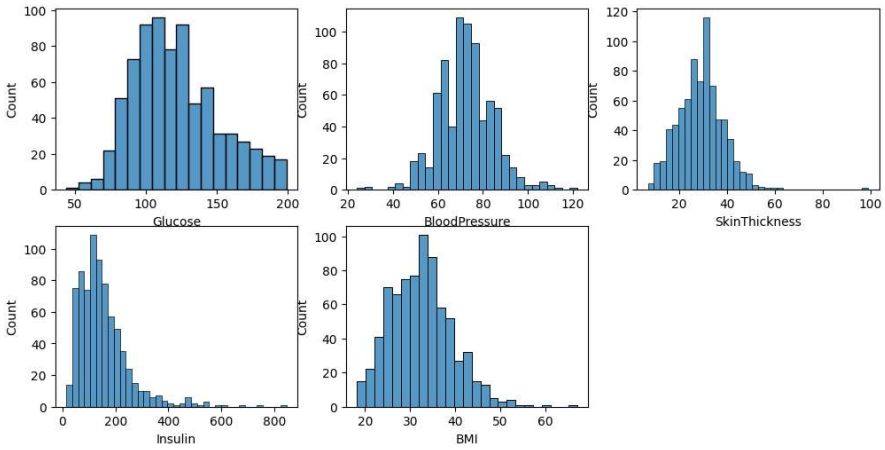
```
df_imputed.head()
```

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	148.0	72.0	35.0	169.0	33.6	0.627	50.0
1	85.0	66.0	29.0	58.6	26.6	0.351	31.0
2	183.0	64.0	23.4	174.6	23.3	0.672	32.0
3	89.0	66.0	23.0	94.0	28.1	0.167	21.0
4	137.0	40.0	35.0	168.0	43.1	2.288	33.0

```
# check the distribution after the missing value treatment
plt.figure(figsize=(12,9))
plt.subplot(3,3,1)
sns.histplot(df_imputed['Glucose'])
plt.subplot(3,3,2)
sns.histplot(df_imputed['BloodPressure'])
plt.subplot(3,3,3)
```

```
sns.histplot(df_imputed['SkinThickness'])
plt.subplot(3,3,4)
sns.histplot(df_imputed['Insulin'])
plt.subplot(3,3,5)
sns.histplot(df_imputed['BMI'])
```

```
<Axes: xlabel='BMI', ylabel='Count'>
```



We can see the distribution of graph is balanced.

```
# Now extract the pregnencies column
Extracted_col = df_train[['Pregnancies']]
```

```
# joining the pregnencies column with the imputed dataset
feature_column = Extracted_col.join(df_imputed)
```

```
feature_column.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	148.0	72.0	35.0	169.0	33.6	
1	1	85.0	66.0	29.0	58.6	26.6	
2	8	183.0	64.0	23.4	174.6	23.3	
3	1	89.0	66.0	23.0	94.0	28.1	
4	0	137.0	40.0	35.0	168.0	43.1	

```
target_column = df_train['Outcome']
```

```
# join the target variable with the feature variables
# our dataframe is cleaned and ready to be used for modelling
train_df = feature_column.join(target_column)
```

```
train_df.head()
```

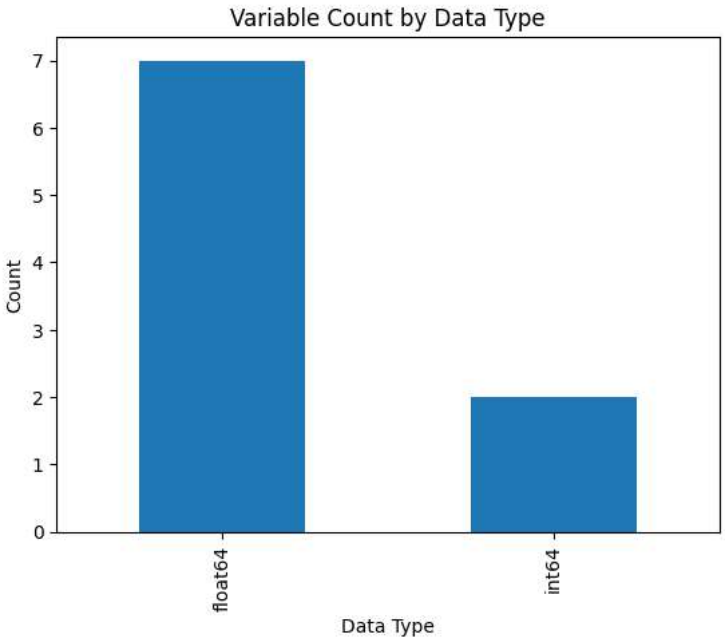
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Ou
2	8	183.0	64.0	23.4	174.6	23.3	0.672	32.0	

```
# import the dataframe for visualization using tableau
 train_df.to_csv('Diabetes.csv')
```

```
#count of variables by datatype
count_dtype = train_df.dtypes.value_counts()
```

4	0	137.0	40.0	35.0	168.0	43.1	2.288	33.0	

```
# create a frequency plot
count_dtype.plot(kind='bar')
plt.title('Variable Count by Data Type')
plt.xlabel('Data Type')
plt.ylabel('Count')
plt.show()
```



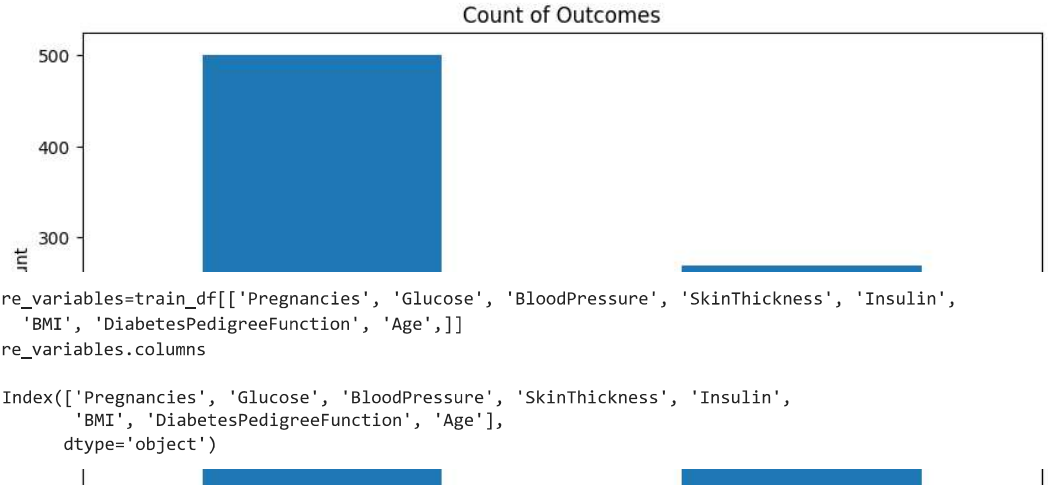
Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.

```
# count the value of outcome column
target_values_count= pd.value_counts(train_df['Outcome'])
target_values_count
```

```
0    500
1    268
Name: Outcome, dtype: int64
```

```
#Plot the frequency plot to check the binary output
plt.figure(figsize=(10,5))
target_values_count.plot(kind='bar')
plt.title('Count of Outcomes')
plt.xlabel('Outcome')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```




```
feature_variables=train_df[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age',]]
feature_variables.columns
```

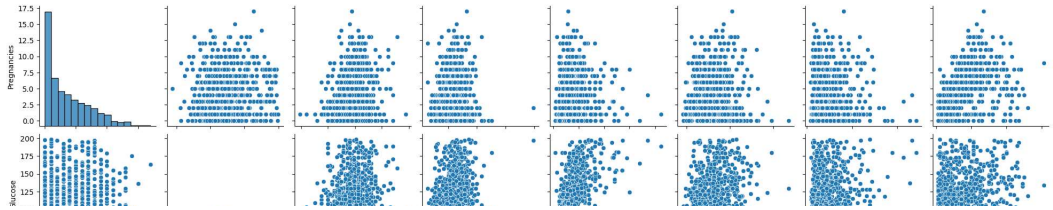
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age'],
      dtype='object')
```


Create scatter charts between the pair of variables to understand the relationships. Describe your findings.

Outcome

```
# using pairplot we plot scatter charts to see the relationship amongst the variables.
sns.pairplot(feature_variables,kind='scatter')
```

```
<seaborn.axisgrid.PairGrid at 0x7fc2ffec83d0>
```

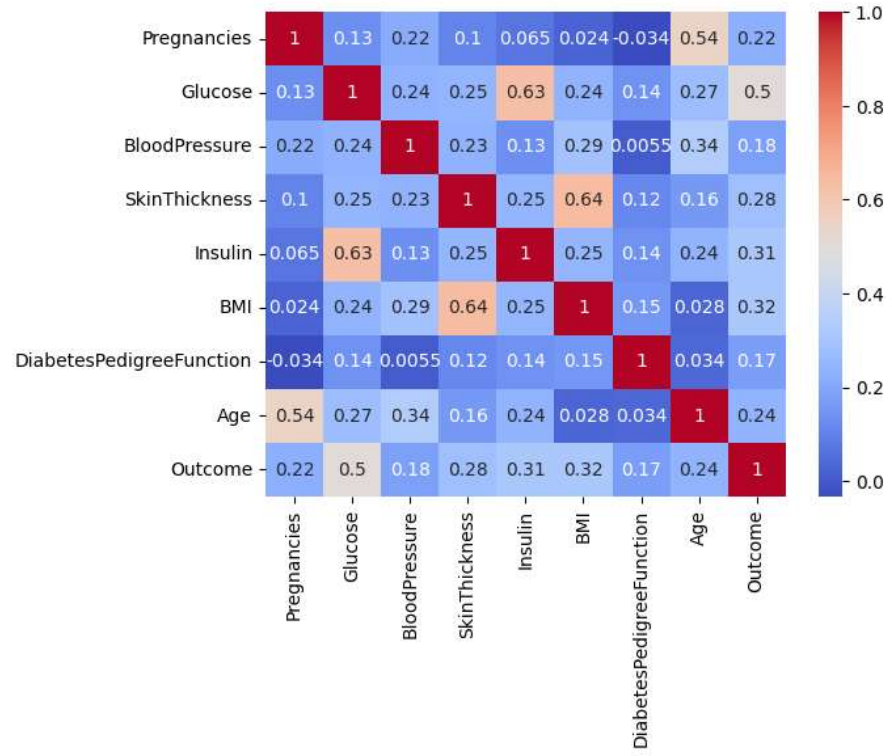


Perform correlation analysis. Visually explore it using a heat map

```
Correlation=train_df.corr()
```

```
sns.heatmap(Correlation, cmap='coolwarm', annot=True)
```

```
<Axes: >
```



Project Task: Week 3 Data Modeling:

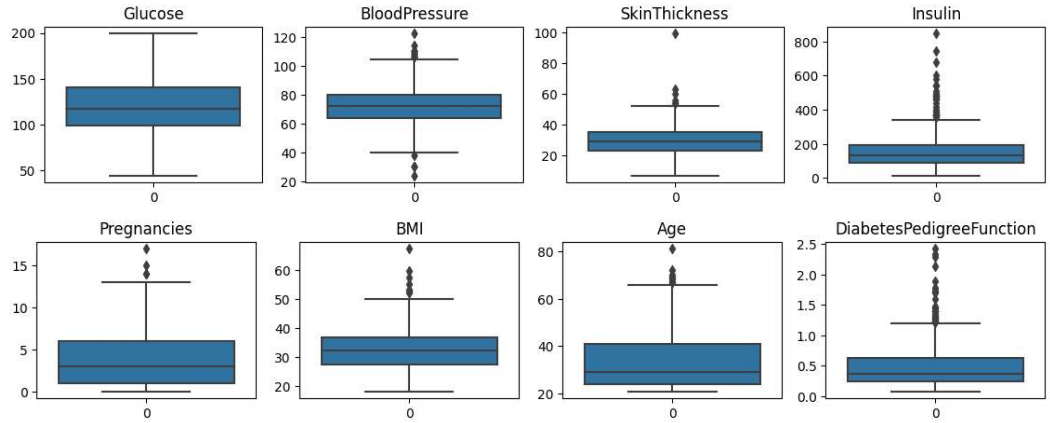
Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.

Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.

```
# Create box plot to see each feature variables
plt.figure(figsize=(14,10))
plt.subplot(4,4,1)
sns.boxplot(train_df['Glucose'])
plt.title('Glucose')
plt.subplot(4,4,2)
sns.boxplot(train_df['BloodPressure'])
plt.title('BloodPressure')
plt.subplot(4,4,3)
sns.boxplot(train_df['SkinThickness'])
plt.title('SkinThickness')
plt.subplot(4,4,4)
sns.boxplot(train_df['Insulin'])
plt.title('Insulin')
plt.figure(figsize=(14,10))
plt.subplot(4,4,5)
sns.boxplot(train_df['Pregnancies'])
plt.title('Pregnancies')
plt.subplot(4,4,6)
sns.boxplot(train_df['BMI'])
```

```
plt.title('BMI')
plt.subplot(4,4,7)
sns.boxplot(train_df['Age'])
plt.title('Age')
plt.subplot(4,4,8)
sns.boxplot(train_df['DiabetesPedigreeFunction'])
plt.title('DiabetesPedigreeFunction')
```

```
Text(0.5, 1.0, 'DiabetesPedigreeFunction')
```




```
# Check and remove outliers from the dataset
def remove_outliers_iqr(feature_variables):
    q1, q3 = np.percentile(feature_variables, [25, 75])
    iqr = q3 - q1
    lower_bound = q1 - (1.5 * iqr)
    upper_bound = q3 + (1.5 * iqr)
    return feature_variables[(feature_variables >= lower_bound) & (feature_variables <= upper_bound)]


df_train_clean = remove_outliers_iqr(feature_variables)


num_outliers = len(feature_variables) - len(df_train_clean)


num_outliers
```

```
0
```

```
# import the libraries for modelling
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


# feature and target
X= feature_variables
y= train_df['Outcome']


# train test split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=10)


# Making model using Logistic regression Algorithm
log_reg = LogisticRegression(max_iter=500)
Model_1=log_reg.fit(X_train,y_train)
Model_1_predict=log_reg.predict(X_test)
print("Logistic Regression accuracy:", accuracy_score(y_test, Model_1_predict))
```

```
Logistic Regression accuracy: 0.7597402597402597
```

```
# making Model using Decision Tree algorithm
dec_tree = DecisionTreeClassifier()
Model_2 = dec_tree.fit(X_train,y_train)
Model_2_predict = dec_tree.predict(X_test)
print("Decision Tree accuracy:",accuracy_score(y_test,Model_2_predict))
```

```
Decision Tree accuracy: 0.6688311688311688
```

```
# Making model using Random Forest Algorithm
ran_for = RandomForestClassifier()
Model_3 = ran_for.fit(X_train,y_train)
Model_3_predict = ran_for.predict(X_test)
print("Random Forest Classifier accuracy:",accuracy_score(y_test,Model_3_predict))
```

```
Random Forest Classifier accuracy: 0.7207792207792207
```

```
# Making model using Support Vector machine Algorithm
sup_vec_m = SVC()
Model_4 = sup_vec_m.fit(X_train,y_train)
Model_4_predict = sup_vec_m.predict(X_test)
print("Support Vector Machine accuracy:",accuracy_score(y_test,Model_4_predict))
```

```
Support Vector Machine accuracy: 0.7077922077922078
```

```
# Making model using K_Nearest neighbour Algorithm
KNN = KNeighborsClassifier()
Model_5 = KNN.fit(X_train,y_train)
Model_5_predict = KNN.predict(X_test)
print("K-Nearest neighbour accuracy:",accuracy_score(y_test,Model_5_predict))
```

```
K-Nearest neighbour accuracy: 0.6558441558441559
```

Data Modeling:

Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used.

```
# Creating Classification report for all the Algorithm
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, roc_auc_score
print("Logistic Regression Classification Report:\n",classification_report(y_test,Model_1_predict))
print("Decision tree Classification Report:\n",classification_report(y_test,Model_2_predict))
print("Random Forest Classification Report:\n",classification_report(y_test,Model_3_predict))
print("Support Vector Machine Classification Report:\n",classification_report(y_test,Model_4_predict))
print("K_Nearest Neighbour Classification Report:\n",classification_report(y_test,Model_5_predict))
```

```
Logistic Regression Classification Report:
               precision    recall  f1-score   support

           0       0.76      0.88      0.82        95
           1       0.75      0.56      0.64        59

    accuracy                           0.76       154
   macro avg       0.76      0.72      0.73       154
weighted avg       0.76      0.76      0.75       154

Decision tree Classification Report:
               precision    recall  f1-score   support

           0       0.72      0.76      0.74        95
           1       0.57      0.53      0.55        59

    accuracy                           0.67       154
   macro avg       0.65      0.64      0.64       154
weighted avg       0.66      0.67      0.67       154

Random Forest Classification Report:
               precision    recall  f1-score   support

           0       0.73      0.87      0.79        95
           1       0.70      0.47      0.57        59

    accuracy                           0.72       154
```

```
           macro avg       0.71      0.67      0.68       154
        weighted avg       0.72      0.72      0.71       154

        Support Vector Machine Classification Report:
                      precision    recall  f1-score   support

                   0       0.70      0.92      0.79        95
                   1       0.73      0.37      0.49        59

            accuracy                           0.71       154
           macro avg       0.72      0.64      0.64       154
        weighted avg       0.71      0.71      0.68       154

        K_Nearest Neighbour Classification Report:
                      precision    recall  f1-score   support

                   0       0.69      0.80      0.74        95
                   1       0.57      0.42      0.49        59

            accuracy                           0.66       154
           macro avg       0.63      0.61      0.61       154
        weighted avg       0.64      0.66      0.64       154
```

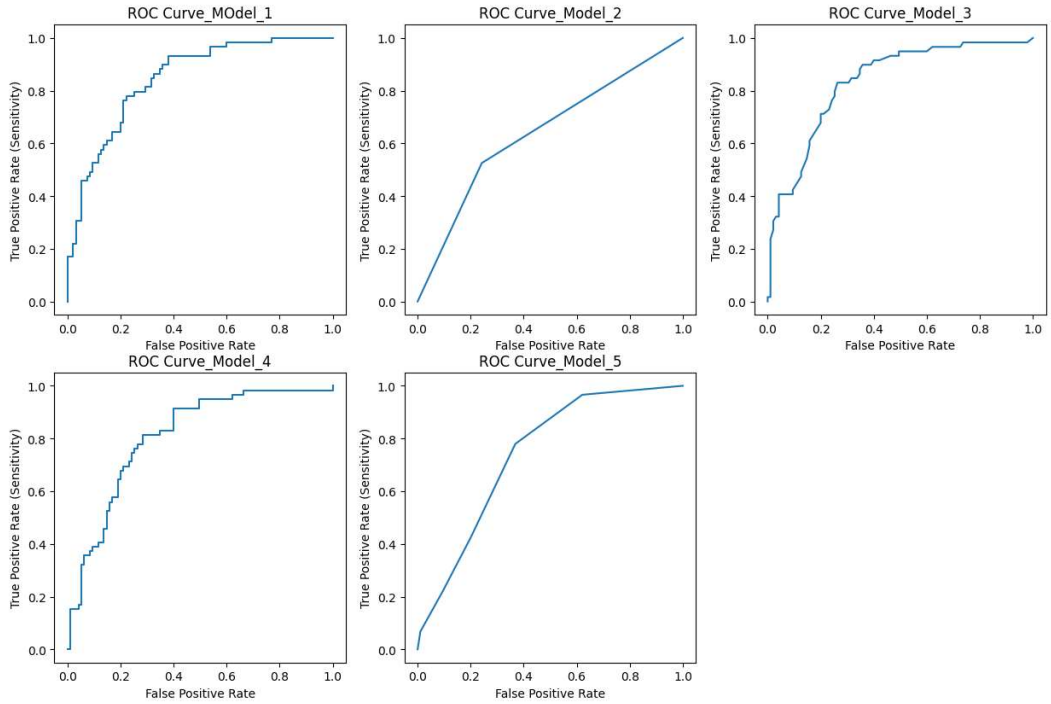
Logistic Regression algorith have the best accuracy and hence, it can be considered as the best model out of five.

```
# Creating Confusion matrix
conf_mat_1 = confusion_matrix(y_test, Model_1_predict)
conf_mat_2 = confusion_matrix(y_test, Model_2_predict)
conf_mat_3 = confusion_matrix(y_test, Model_3_predict)
conf_mat_4 = confusion_matrix(y_test, Model_4_predict)
conf_mat_5 = confusion_matrix(y_test, Model_5_predict)
print("Logistic Regression Confusion Matrix:\n", conf_mat_1)
print("Decision tree Confusion Matrix:\n", conf_mat_2)
print("Random Forest Confusion Matrix:\n", conf_mat_3)
print("Support Vector Machine Confusion Matrix:\n", conf_mat_4)
print("K-Nearest Neighbour Confusion Matrix:\n", conf_mat_5)
```

```
Logistic Regression Confusion Matrix:
 [[84 11]
 [26 33]]
Decision tree Confusion Matrix:
 [[72 23]
 [28 31]]
Random Forest Confusion Matrix:
 [[83 12]
 [31 28]]
Support Vector Machine Confusion Matrix:
 [[87  8]
 [37 22]]
K-Nearest Neighbour Confusion Matrix:
 [[76 19]
 [34 25]]
```

```
# Generate an ROC curve
plt.figure(figsize=(15,15))
fpr, tpr, thresholds = roc_curve(y_test, log_reg.decision_function(X_test))
plt.subplot(3,3,1)
plt.plot(fpr, tpr, label="ROC Curve_Model_1")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate (Sensitivity)")
plt.title("ROC Curve_MOdel_1")
plt.subplot(3,3,2)
fpr, tpr, thresholds = roc_curve(y_test, dec_tree.predict_proba(X_test)[:,1])
plt.plot(fpr, tpr, label="ROC Curve_Model_2")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate (Sensitivity)")
plt.title("ROC Curve_Model_2")
plt.subplot(3,3,3)
fpr, tpr, thresholds = roc_curve(y_test, ran_for.predict_proba(X_test)[:,1])
plt.plot(fpr, tpr, label="ROC Curve_Model_3")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate (Sensitivity)")
plt.title("ROC Curve_Model_3")
plt.subplot(3,3,4)
fpr, tpr, thresholds = roc_curve(y_test, sup_vec_m.decision_function(X_test))
plt.plot(fpr, tpr, label="ROC Curve_Model_4")
plt.xlabel("False Positive Rate")
```

```
plt.ylabel("True Positive Rate (Sensitivity)")
plt.title("ROC Curve_Model_4")
plt.subplot(3,3,5)
fpr, tpr, thresholds = roc_curve(y_test, KNN.predict_proba(X_test)[:,1])
plt.plot(fpr, tpr, label="ROC Curve_Model_5")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate (Sensitivity)")
plt.title("ROC Curve_Model_5")
plt.show()
```




```
# Calculate the AUC score
auc_Model_1 = roc_auc_score(y_test, log_reg.decision_function(X_test))
print("Logistic Regression AUC Score:", auc_Model_1)
auc_Model_2 = roc_auc_score(y_test, dec_tree.predict_proba(X_test)[:,1])
print("Decision Tree AUC Score:", auc_Model_2)
auc_Model_3 = roc_auc_score(y_test, ran_for.predict_proba(X_test)[:,1])
print("Random Forest AUC Score:", auc_Model_3)
auc_Model_4 = roc_auc_score(y_test, sup_vec_m.decision_function(X_test))
print("Support Vector Mahine AUC Score:", auc_Model_4)
auc_Model_5 = roc_auc_score(y_test, KNN.predict_proba(X_test)[:,1])
print("K-Nearest neighbour AUC Score:", auc_Model_5)
```

```
Logistic Regression AUC Score: 0.8456735057983943
Decision Tree AUC Score: 0.6416592328278323
Random Forest AUC Score: 0.8327386262265833
Support Vector Mahine AUC Score: 0.808385370205174
K-Nearest neighbour AUC Score: 0.7407671721677074
```

Overall we can conclude that Logistic Regression algorithm is giving the best result and hence it can be used for deployment.

Visualization-Tableau dashboard-- link-https://public.tableau.com/app/profile/lily.priyadrashini/viz/HealthcareDiabeticNon-Diabetic/DiabeticNon-DiabeticAnalysis

✓ 0s completed at 5:20 PM

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.